

## PLANIFICACIÓN DE PROYECTOS DE SOFTWARE

**L**A gestión de un proyecto de software comienza con un conjunto de actividades que globalmente se denominan *planificación del proyecto*. Antes de que el proyecto comience, el gestor y el equipo de software deben realizar una estimación del trabajo a realizar, de los recursos necesarios y del tiempo que transcurrirá desde el comienzo hasta el final de su realización. Siempre que estimamos, estamos mirando hacia el futuro y aceptamos resignados cierto grado de incertidumbre.

Aunque la estimación es más un arte que una ciencia, es una actividad importante que no debe llevarse a cabo de forma descuidada. Existen técnicas útiles para la estimación del esfuerzo y del tiempo. Las métricas del proyecto y del proceso proporcionan una perspectiva histórica y una potente introducción para generar estimaciones cuantitativas. La experiencia anterior (de todas las personas involucradas) puede ayudar en gran medida al desarrollo y revisión de las estimaciones. Y, dado que la estimación es la base de todas las demás actividades de planificación del proyecto, y que sirve como guía para una buena ingeniería del software, no es en absoluto aconsejable embarcarse sin ella.

## VISTAZO RÁPIDO

**¿Qué es?** La planificación de un proyecto de software realmente comprende todas las actividades tratadas en los Capítulos 5 al 9. Sin embargo, en el contexto de este capítulo, la planificación implica la estimación —su intento por determinar cuánto dinero, esfuerzo, recursos, y tiempo supondrá construir un sistema o producto específico de software—.

**¿Quién lo hace?** Los gestores del software —utilizando la información solicitada a los clientes y a los ingenieros de software y los datos de métricas de software obtenidos de proyectos anteriores—.

**¿Por qué es importante?** ¿Podría construir una casa sin saber cuánto estaría dispuesto a gastar?. Por supuesto que no, y puesto que la mayoría de los siste-

mas y productos basados en computadora cuestan considerablemente más que construir una casa grande, podría ser razonable desarrollar y estimar antes de empezar a construir el software.

**¿Cuáles son los pasos?** La estimación comienza con una descripción del ámbito del producto. Hasta que no se «delimita» el ámbito no es posible realizar una estimación con sentido. El problema es entonces descompuesto en un conjunto de problemas de menor tamaño y cada uno de éstos se estima guiándose con datos históricos y con la experiencia. Es aconsejable realizar las estimaciones utilizando al menos dos métodos diferentes (como comprobación). La complejidad y el riesgo del problema se considera antes de realizar una estimación final.

**¿Cuál es el producto obtenido?** Se obtiene una tabla que indica las tareas a desarrollar, las funciones a implementar, y el coste, esfuerzo y tiempo necesario para la realización de cada una. También se obtiene una lista de recursos necesarios para el proyecto.

**¿Cómo puedo estar seguro de que lo he hecho correctamente?** Esto es difícil, puesto que realmente no lo sabrá hasta que el proyecto haya finalizado. Sin embargo, si tiene experiencia y sigue un enfoque sistemático, realiza estimaciones utilizando datos históricos sólidos, crea puntos de estimación mediante al menos dos métodos diferentes y descompone la complejidad y riesgo, puede estar seguro de haber acertado plenamente.

## 5.1 OBSERVACIONES SOBRE LA ESTIMACIÓN

A un destacado ejecutivo se le preguntó una vez por la característica más importante que debe tener un gestor de proyectos. Respondió: «...una persona con la habilidad de saber qué es lo que va a ir mal antes de que ocurra...». Debemos añadir: «...y con el coraje para hacer estimaciones cuando el futuro no está claro...».

La estimación de recursos, costes y planificación temporal de un esfuerzo en el desarrollo de software requiere experiencia, acceder a una buena información histórica y el coraje de confiar en predicciones (medidas) cuantitativas cuando todo lo que existe son datos cualitativos. La estimación conlleva un riesgo inherente<sup>1</sup> y es este riesgo el que lleva a la incertidumbre.



**Cita:**

Buenos enfoques de estimación y datos históricos sólidos ofrecen la mejor esperanza de que la realidad puede vencer sobre las demandas imposibles.

Capers Jones

La complejidad del proyecto tiene un gran efecto en la incertidumbre, que es inherente en la planificación. Sin embargo, la complejidad es una medida relativa que se ve afectada por la familiaridad con esfuerzos anteriores. Se podría considerar una aplicación sofisticada de comercio electrónico como «excesivamente compleja» para un desarrollador que haya realizado su primera aplicación. Sin embargo para un equipo de software que desarrolle su enésimo sitio web de comercio electrónico podría considerarse «sumamente fácil» (una de tantas). Se han propuesto una serie de medidas cuantitativas de la complejidad del software (por ejemplo, [ZU597]). Tales medidas se aplican en el nivel de diseño y de codificación, y por consiguiente son difíciles de utilizar durante la planificación del software (antes de que exista un diseño o un código). Sin embargo, al comienzo del proceso de planificación se pueden establecer otras valoraciones de complejidad más subjetivas (por ejemplo, los factores de ajuste de la complejidad del punto de función descritos en el Capítulo 4).

El tamaño del proyecto es otro factor importante que puede afectar a la precisión y a la eficiencia de las estimaciones. A medida que el tamaño aumenta, crece rápidamente<sup>2</sup> la interdependencia entre varios elementos del software. El problema de la descomposición, un enfoque importante hacia la estimación, se hace más difícil porque los elementos descompuestos pueden ser todavía excesivamente grandes. Parafraseando la ley de Murphy: «lo que puede ir mal irá mal», y si hay más cosas que puedan fallar, más cosas fallarán.

<sup>1</sup> En el Capítulo 6 se presentan técnicas sistemáticas para el análisis del riesgo.

<sup>2</sup> El tamaño se incrementa con frecuencia debido al cambio del ámbito que ocurre cuando el cliente modifica los requisitos. El incremento del tamaño del proyecto puede tener un impacto geométrico en el coste y en la planificación del proyecto [MAH96].



**CLAVE**

La complejidad del proyecto, el tamaño del proyecto y el grado de incertidumbre estructural afectan a la fiabilidad de la estimación.

El grado de incertidumbre estructural tiene también efecto en el riesgo de la estimación. En este contexto, la estructura se refiere al grado en el que los requisitos se han definido, la facilidad con la que pueden subdividirse funciones, y la naturaleza jerárquica de la información que debe procesarse.

La disponibilidad de información histórica tiene una fuerte influencia en el riesgo de la estimación. Al mirar atrás, podemos emular lo que se ha trabajado y mejorar las áreas en donde surgieron problemas. Cuando se dispone de las métricas completas de software de proyectos anteriores (Capítulo 4), se pueden hacer estimaciones con mayor seguridad; establecer planificaciones para evitar dificultades anteriores, y así reducir el riesgo global.



**Cita:**

Lo que caracteriza a una inteligencia formada es que puede descansar satisfecha con el grado de precisión que la naturaleza de un asunto permite, y no buscar la exactitud cuando sólo una aproximación de la verdad es posible...

Aristóteles

El riesgo se mide por el grado de incertidumbre en las estimaciones cuantitativas establecidas por recursos, coste y planificación temporal. Si no se entiende bien el ámbito del proyecto o los requisitos del proyecto están sujetos a cambios, la incertidumbre y el riesgo son peligrosamente altos. El planificador del software debería solicitar definiciones completas de rendimiento y de interfaz (dentro de una especificación del sistema). El planificador y, lo que es más importante, el cliente, deben tener presente que cualquier cambio en los requisitos del software significa inestabilidad en el coste y en la planificación temporal.

Sin embargo, un gestor de proyecto no debería obsesionarse con la estimación. Los enfoques modernos de ingeniería del software (por ejemplo, modelos de procesos evolutivos) toman un punto de vista iterativo del desarrollo. En tales enfoques, es posible<sup>3</sup> revisar la estimación (a medida que se conoce más información), y variarla cuando el cliente haga cambios de requisitos.

<sup>3</sup> Esto no significa que siempre sea aceptable políticamente modificar las estimaciones iniciales. Una organización de software madura y sus gestores reconocen que el cambio no es libre. Y sin embargo muchos clientes solicitan (incorrectamente) que una vez realizada la estimación debería mantenerse independientemente de que las circunstancias cambien.

## 5.2 OBJETIVOS DE LA PLANIFICACIÓN DEL PROYECTO

El objetivo de la planificación del proyecto de software es proporcionar un marco de trabajo que permita al gestor hacer estimaciones razonables de recursos, coste y planificación temporal. Estas estimaciones se hacen dentro de un marco de tiempo limitado al comienzo de un proyecto de software, y deberían actualizarse regularmente a medida que progresa el proyecto. Además, las estimaciones deberían definir los escenarios del «mejor caso» y «peor caso» de forma que los resultados del proyecto puedan limitarse.

El objetivo de la planificación se logra mediante un proceso de descubrimiento de la información que lleve a estimaciones razonables. En las secciones siguientes, se estudian cada una de las actividades asociadas a la planificación del proyecto de software.



*Cuanto más sepa, mejor realizará la estimación.  
Por consiguiente, actualice sus estimaciones  
a medida que progresa el proyecto.*

## 5.3 ÁMBITO DEL SOFTWARE

La primera actividad de la planificación del proyecto de software es determinar el ámbito del software. Se deben evaluar la función y el rendimiento que se asignaron al software durante la ingeniería del sistema de computadora (Capítulo 10), para establecer un ámbito de proyecto que no sea ambiguo, ni incomprensible para directivos y técnicos. Se debe *delimitar* la declaración del ámbito del software.

El *ámbito del software* describe el control y los datos a procesar, la función, el rendimiento, las restricciones, las interfaces y la fiabilidad. Se evalúan las funciones descritas en la declaración del ámbito, y en algunos casos se refinan para dar más detalles antes del comienzo de la estimación. Dado que las estimaciones del coste y de la planificación temporal están orientadas a la función, muchas veces es útil llegar a un cierto grado de descomposición. Las consideraciones de rendimiento abarcan los requisitos de tiempo de respuesta y de procesamiento. Las restricciones identifican los límites del software originados por el hardware externo, por la memoria disponible y por otros sistemas existentes.

### 5.3.1. Obtención de la información necesaria para el ámbito

Al principio de un proyecto de software las cosas siempre están un poco borrosas. Se ha definido una necesidad y se han enunciado las metas y objetivos básicos, **pero** todavía no se ha establecido la información necesaria para definir el ámbito (prerrequisito para la estimación).

La técnica utilizada con más frecuencia para acercarse al cliente y al desarrollador, y para hacer que comience el proceso de comunicación es establecer una reunión o una entrevista preliminar. La primera reunión entre un ingeniero de software (el analista) y el cliente puede compararse a la primera cita entre adolescentes. Ninguna persona sabe lo que decir o preguntar; ambos están preocupados por si lo que dicen es mal interpre-

tado; ambos están pensando hasta dónde podrían llegar (probablemente los dos tienen aquí diferentes expectativas); ambos quieren quitárselo pronto de encima; pero al mismo tiempo quieren que salga bien.



**¿Cómo debería empezar la comunicación entre el desarrollador y el cliente?**

Sin embargo, se debe iniciar la comunicación. Gause y Weinberg [GAU89] sugieren que el analista comience haciendo *preguntas de contexto libre*. Es decir, una serie de preguntas que lleven a un entendimiento básico del problema, las personas que están interesadas en la solución, la naturaleza de la solución que se desea y la efectividad prevista del primer encuentro.

El primer conjunto de cuestiones de contexto libre se centran en el cliente, en los objetivos globales y en los beneficios. Por ejemplo, el analista podría preguntar:

- ¿Quién está detrás de la solicitud de este trabajo?
- ¿Quién utilizará la solución?
- ¿Cuál será el beneficio económico de una buena solución?
- ¿Hay otro camino para la solución?

Las preguntas siguientes permiten que el analista comprenda mejor el problema y que el cliente exprese sus percepciones sobre una solución:

- ¿Cómo caracterizaría [el cliente] un resultado «correcto» que se generaría con una solución satisfactoria?
- ¿Con qué problema(s) se afrontará esta solución?
- ¿Puede mostrarme (o describirme) el entorno en el que se utilizará la solución?
- ¿Hay aspectos o limitaciones especiales de rendimiento que afecten a la forma en que se aborde la solución?

La última serie de preguntas se centra en la efectividad de la reunión. Gause y Weinberg las llaman «meta-cuestiones» y proponen la lista siguiente (abreviada):

- ¿Es usted la persona apropiada para responder a estas preguntas? ¿Son «oficiales» sus respuestas?
- ¿Son relevantes mis preguntas para su problema?
- ¿Estoy realizando muchas preguntas?
- ¿Hay alguien más que pueda proporcionar información adicional?
- ¿Hay algo más que debiera preguntarle?

Estas preguntas (y otras) ayudarán a «romper el hielo» y a iniciar la comunicación esencial para establecer el ámbito del proyecto. Sin embargo, una reunión basada en preguntas y respuestas no es un enfoque que haya tenido un éxito abrumador. En realidad, la sesión P&R sólo se debería utilizar para el primer encuentro, reemplazándose posteriormente por un tipo de reunión que combine elementos de resolución de problemas, negociación y especificación.

Los clientes y los ingenieros de software con frecuencia tienen establecido inconscientemente el pensamiento de «nosotros y ellos». En lugar de trabajar como un equipo para identificar y refinar los requisitos, cada uno define su propio «territorio» y se comunica por medio de memorandos, documentos formales de situación, sesiones de preguntas y respuestas e informes. La historia ha demostrado que este enfoque es muy pobre. Abundan los malentendidos, se omite información importante y nunca se establece una relación de trabajo con éxito.

#### Referencia cruzada

Las técnicas de obtención de requisitos se tratan en el Capítulo 11.

Con estos problemas en mente un grupo de investigadores independientes ha desarrollado un enfoque orientado al equipo para la recopilación de requisitos que pueden aplicarse para ayudar a establecer el ámbito de un proyecto. Las técnicas denominadas *técnicas para facilitar las especificaciones de la aplicación* (TFEA) (*facilitated application specification techniques* [FAST]), pertenecen a un enfoque que alienta a la creación de un equipo compuesto de clientes y de desarrolladores que trabajen juntos para identificar el problema, proponer elementos de solución, negociar diferentes enfoques y especificar un conjunto preliminar de requisitos.

### 5.3.2. Viabilidad

Una vez se ha identificado el ámbito (con la ayuda del cliente), es razonable preguntarse: «¿Podemos construir el software de acuerdo a este ámbito? ¿Es factible el proyecto?». Con frecuencia, las prisas de los ingenieros de software sobrepasan estas preguntas (o están obligados a pasarlas por los clientes o gestores impacientes), solo se tienen en cuenta en un proyecto condenado desde el comienzo. Putnam y Myers [PUT97a] tratan este aspecto cuando escriben:

#### Cita:

Hay 150 km a Chicago, tenemos lleno el tanque de gasolina, medio paquete de cigarrillos, está oscuro y llevamos gafos de sol. ¡Vamos!  
The Blues Brothers

...no todo lo imaginable es factible, ni siquiera en el software, fugaz como puede aparecer un forastero. Por el contrario la factibilidad del software tiene cuatro dimensiones sólidas: *Tecnología*—¿Es factible un proyecto técnicamente? ¿Está dentro del estado actual de la técnica? *Financiación*—¿Es factible financieramente? ¿Puede realizarse a un coste asumible por la empresa de software y por el cliente? *Tiempo*—¿Pueden los proyectos adelantarse a los de la competencia? *Recursos*—¿La organización cuenta con los recursos suficientes para tener éxito?

La respuesta es sencilla para algunos proyectos en ciertas áreas, ya que se ha hecho antes algún proyecto de este tipo. A las pocas horas, o, a veces, en pocas semanas de investigación, se puede estar seguro que se puede hacer de nuevo.

Los proyectos de los que no se tiene experiencia son fáciles. Un equipo puede pasarse varios meses descubriendo cuáles son los requisitos principales, y cuáles son aquellos difíciles de implementar para una nueva aplicación. ¿Podría ocurrir que alguno de estos requerimientos presentara algunos riesgos que hicieran inviable el proyecto? ¿Podrían superarse estos riesgos? El equipo de factibilidad debe asumir la arquitectura y el diseño de los requerimientos de alto riesgo hasta el punto de poder responder todas estas preguntas. En algunos casos, cuando el equipo proporciona respuestas negativas, esto puede negociarse con una reducción de los requisitos.

Mientras tanto, los altos ejecutivos están repicando sus dedos en la mesa. A menudo, mueven sus cigarrillos de forma elegante, pero de forma impaciente y rodeados de humo exclaman ¡Ya es suficiente! ¡Hazlo!

Muchos de estos proyectos que se han aprobado de esta forma aparecen a los pocos años en la prensa como proyectos defectuosos.

#### CONSEJO

El estudio de viabilidad es importante, pero las necesidades de la gestión son incluso más importantes. No es bueno construir un producto o sistema de alta tecnología que en realidad nadie quiera.

Putnam y Myers sugieren, de forma acertada, que el estudio del ámbito no es suficiente. Una vez que se ha comprendido el ámbito, tanto el equipo de desarrollo como el resto deben trabajar para determinar si puede ser construido dentro de las dimensiones reflejadas anteriormente. Esto es crucial, aunque es una parte del proceso de estimación pasada por alto a menudo.

### 5.3.3. Un ejemplo de ámbito

La comunicación con el cliente lleva a una definición del control y de los datos procesados, de las funciones que deben ser implementadas, del rendimiento y restricciones que delimitan el sistema, y de la información relacionada. Como ejemplo, consideremos el software

que se debe desarrollar para controlar un sistema de clasificación de cinta transportadora (SCCT). La especificación del ámbito del SCCT es la siguiente:

*El sistema de clasificación de cinta transportadora (SCCT) clasifica las cajas que se mueven por una cinta transportadora. Cada caja estará identificada por un código de barras que contiene un número de pieza y se clasifica en uno de seis compartimentos al final de la cinta. Las cajas pasarán por una estación de clasificación que consta de un lector de código de barras y un PC. El PC de la estación de clasificación está conectado a un mecanismo de maniobra que clasifica las cajas en los compartimentos. Las cajas pasan en orden aleatorio y están espaciadas uniformemente. La cinta se mueve a cinco pies por minuto. En la Figura 5.1 está representado esquemáticamente el SCCT.*

El software del SCCT debe recibir información de entrada de un lector de código de barras a intervalos de tiempo que se ajusten a la velocidad de la cinta transportadora. Los datos del código de barras se decodifican al formato de identificación de caja. El software llevará a cabo una inspección en la base de datos de números de piezas que contiene un máximo de 1000 entradas para determinar la posición del compartimento adecuada para la caja que se encuentre actualmente en el lector (estación de clasificación). La posición correcta del compartimento se pasará a un mecanismo de maniobra de ordenación que sitúa las cajas en el lugar adecuado. Se mantendrá una lista con los compartimentos destino de cada caja para su posterior recuperación e informe. El software del SCCT recibirá también entrada de un tacómetro de pulsos que se utilizará para sincronizar la señal de control del mecanismo de maniobra. Basándonos en el número de pulsos que se generen entre la estación de clasificación y el mecanismo de maniobra, el software producirá una señal de control para que la maniobra sitúe adecuadamente la caja.

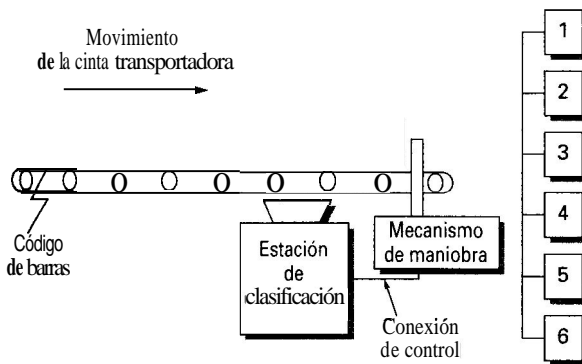


FIGURA 5.1. Un sistema de clasificación de cinta transportadora.

El planificador del proyecto examina la especificación del ámbito y extrae todas las funciones importantes del software. Este proceso, denominado *descomposición*, se trató en el Capítulo 3 y produce como resultado las funciones siguientes<sup>4</sup>:

- Lectura de la entrada del código de barras.
- Lectura del tacómetro de pulsos.
- Descodificación de los datos del código de pieza.

<sup>4</sup> En realidad, la descomposición funcional se hace durante la ingeniería del sistema (Capítulo 10). El planificador utiliza la información obtenida a partir de la especificación del sistema para definir funciones del software.

- Búsqueda en la base datos.
- Determinar la posición del compartimento.
- Producción de la señal de control para el mecanismo de maniobra.
- Mantener una lista de los destinos de las cajas

En este caso, el rendimiento está determinado por la velocidad de la cinta transportadora. Se tiene que terminar el procesamiento de cada caja antes de que llegue la siguiente caja al lector de código de barras. El software del SCCT está limitado por el hardware al que tiene que acceder —el lector de código de barras, el mecanismo de maniobra, la computadora personal (PC)—, la memoria disponible y la configuración global de la cinta transportadora (cajas uniformemente espaciadas).



*Ajuste la estimación para reflejar los requisitos del rendimiento y las restricciones del diseño difíciles, incluso si el ámbito es sencillo.*

La función, el rendimiento y las restricciones se evalúan a la vez. Una misma función puede producir una diferencia de un orden de magnitud en el esfuerzo de desarrollo cuando se considera en un contexto con diferentes límites de rendimiento. El esfuerzo y los costes requeridos para desarrollar el software SCCT serían drásticamente diferentes si la función fuera la misma (por ejemplo: la cinta transportadora siguiese colocando cajas en contenedores), pero el rendimiento variará. Por ejemplo, si la velocidad media de la cinta transportadora aumentara en un factor de 10 (rendimiento) y las cajas no estuvieran uniformemente espaciadas (una restricción), el software podría ser considerablemente más complejo —requiriendo, por tanto, un mayor esfuerzo de desarrollo—. La función, el rendimiento y las restricciones están íntimamente relacionadas.

## CLAVE

La consideración del ámbito del software debe contener una evaluación de todas las interfaces externas.

El software interactúa con otros elementos del sistema informático. El planificador considera la naturaleza y la complejidad de cada interfaz para determinar cualquier efecto sobre los recursos, los costes y la planificación temporal del desarrollo. El concepto de interfaz abarca lo siguiente: (1) hardware (por ejemplo: procesador, periféricos) que ejecuta el software y los dispositivos (por ejemplo: máquinas, pantallas) que están controlados indirectamente por el software; (2) softwa-



re ya existente (por ejemplo, rutinas de acceso a una base de datos, componentes de software reutilizables, sistemas operativos) que debe ser integrado con el nuevo software; (3) personas que hacen uso del software por medio del teclado (terminales) u otros dispositivos de entrada/salida; (4) procedimientos que preceden o suceden al software en una secuencia de operaciones. En cada caso, debe comprenderse claramente la información que se transfiere a través de la interfaz.

## 5.4 RECURSOS

La segunda tarea de la planificación del desarrollo de software es la estimación de los recursos requeridos para acometer el esfuerzo de desarrollo de software. La Figura 5.2 ilustra los recursos de desarrollo en forma de pirámide. En la base de la pirámide de recursos se encuentra el *entorno de desarrollo* —herramientas de hardware y software— que proporciona la infraestructura de soporte al esfuerzo de desarrollo. En un nivel más alto se encuentran los *componentes de software reutilizables* —los bloques de software que pueden reducir drásticamente los costes de desarrollo y acelerar la entrega—. En la parte más alta de la pirámide está el recurso primario —*el personal*—. Cada recurso queda especificado mediante cuatro características: descripción del recurso, informe de disponibilidad, fecha cronológica en la que se requiere el recurso, tiempo durante el que será aplicado el recurso. Las dos últimas características pueden verse como una ventana temporal. La disponibilidad del recurso para una ventana específica tiene que establecerse lo más pronto posible.

**?** ¿Cuál es la fuente de información principal para determinar el ámbito?

### 5.4.1. Recursos humanos

El encargado de la planificación comienza elevando el ámbito y seleccionando las habilidades que se requieren para llevar a cabo el desarrollo. Hay que especificar tanto la posición dentro de la organización (por ejemplo: gestor, ingeniero de software experimentado, etc.) como la especialidad (por ejemplo: telecomunicaciones, bases de datos, cliente/servidor). Para proyectos relativamente pequeños (una persona-año o menos) una sola persona puede llevar a cabo todos los pasos de ingeniería del software, consultando con especialistas siempre que sea necesario.

El número de personas requerido para un proyecto de software sólo puede ser determinado después de hacer una estimación del esfuerzo de desarrollo (por ejemplo, personas-mes). Estas técnicas de estimación del esfuerzo se estudiarán después en este mismo capítulo.

Si se ha desarrollado adecuadamente la *especificación del sistema* (véase el Capítulo 10), casi toda la información requerida para la descripción del ámbito del software estará disponible y documentada antes de que comience la planificación del proyecto de software. En los casos en los que no haya sido desarrollada la especificación, el planificador debe hacer el papel del analista de sistemas para determinar las características y las restricciones que influirán en las tareas de estimación.

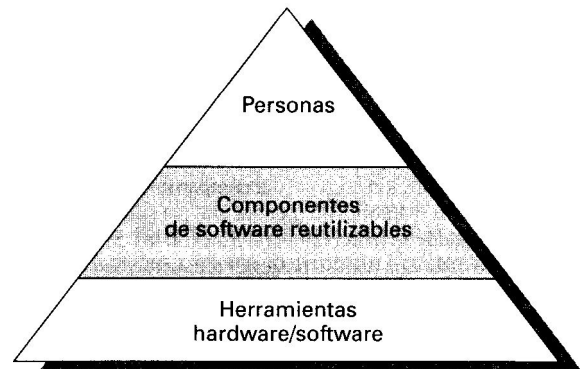


FIGURA 5.2. Recursos del proyecto.

### 5.4.2. Recursos de software reutilizables

La ingeniería del software basada en componentes (ISBC)<sup>5</sup> destaca la reutilización —esto es, la creación y la reutilización de bloques de construcción de software [HOO91]—. Dichos bloques de construcción, llamados componentes, deben establecerse en catálogos para una consulta más fácil, estandarizarse para una fácil aplicación y validarse para una fácil integración.

#### Referencia cruzada

El papel que juegan las personas implicadas en el software y las organizaciones del equipo al que pertenecen se tratan en el Capítulo 3.

Bennatan [BEN92] sugiere cuatro categorías de recursos de software que se deberían tener en cuenta a medida que se avanza con la planificación:

**Componentes ya desarrollados.** El software existente se puede adquirir de una tercera parte o provenir de uno desarrollado internamente para un proyecto anterior. Llamados componentes CCYD (componentes comercialmente ya desarrollados), estos componentes están listos para utilizarse en el proyecto actual y se han validado totalmente.

**Componentes ya experimentados.** Especificaciones, diseños, código o datos de prueba existentes

<sup>5</sup> La ingeniería del software basada en componentes se trata con detalle en el Capítulo 27.

desarrollados para proyectos anteriores que son similares al software que se va a construir para el proyecto actual. Los miembros del equipo de software actual ya han tenido la experiencia completa en el área de la aplicación representada para estos componentes. Las modificaciones, por tanto, requeridas para componentes de total experiencia, tendrán un riesgo relativamente bajo.

### PUNTO CLAVE

Para que la reutilización sea eficiente, los componentes de software deben estar catalogados, estandarizados y validados.

**Componentes con experiencia parcial.** Especificaciones, diseños, código o datos de prueba existentes desarrollados para proyectos anteriores que se relacionan con el software que se va a construir para el proyecto actual, pero que requerirán una modificación sustancial. Los miembros del equipo de software actual han limitado su experiencia sólo al área de aplicación representada por estos componentes. Las modificaciones, por tanto, requeridas para componentes de experiencia parcial tendrán bastante grado de riesgo.

**Componentes nuevos.** Los componentes de software que el equipo de software debe construir específicamente para las necesidades del proyecto actual.

### ¿Qué aspectos debemos considerar cuando planificamos la reutilización de componentes de software existentes?

Deberían ser consideradas las directrices siguientes por el planificador de software cuando los componentes reutilizables se especifiquen como recurso:

1. Si los componentes ya desarrollados cumplen los requisitos del proyecto, adquiéralos. El coste de la adquisición y de la integración de los componentes ya desarrollados serán casi siempre menores que el coste para desarrollar el software equivalente<sup>6</sup>. Además, el riesgo es relativamente bajo.
2. Si se dispone de componentes ya experimentados, los riesgos asociados a la modificación y a la integración

generalmente se aceptan. El plan del proyecto debería reflejar la utilización de estos componentes.

3. Si se dispone de componentes de experiencia parcial para el proyecto actual, su uso se debe analizar con detalle. Si antes de que se integren adecuadamente los componentes con otros elementos del software se requiere una gran modificación, proceda cuidadosamente - e l riesgo es alto—. El coste de modificar los componentes de experiencia parcial algunas veces puede ser mayor que el coste de desarrollar componentes nuevos.

De forma irónica, a menudo se descuida la utilización de componentes de software reutilizables durante la planificación, llegando a convertirse en la preocupación primordial durante la fase de desarrollo del proceso de software. Es mucho mejor especificar al principio las necesidades de recursos del software. De esta forma se puede dirigir la evaluación técnica de alternativas y puede tener lugar la adquisición oportuna.

### 5.4.3. Recursos de entorno

El entorno es donde se apoya el proyecto de software, llamado a menudo *entorno de ingeniería del software (EIS)*, incorpora hardware y software. El hardware proporciona una plataforma con las herramientas (software) requeridas para producir los productos que son el resultado de una buena práctica de la ingeniería del software<sup>7</sup>. Como la mayoría de las organizaciones de software tienen muchos aspectos que requieren acceso a EIS, un planificador de proyecto debe determinar la ventana temporal requerida para el hardware y el software, y verificar que estos recursos estarán disponibles.

Cuando se va a desarrollar un sistema basado en computadora (que incorpora hardware y software especializado), el equipo de software puede requerir acceso a los elementos en desarrollo por otros equipos de ingeniería. Por ejemplo, el software para un *control numérico (CN)* utilizado en una clase de máquina herramienta puede requerir una máquina herramienta específica (por ejemplo, el CN de un torno) como parte del paso de prueba de validación; un proyecto de software para el diseño de páginas avanzado puede necesitar un sistema de composición fotográfica o escritura digital en alguna fase durante el desarrollo. Cada elemento de hardware debe ser especificado por el planificador del proyecto de software.

<sup>6</sup> Cuando los componentes de software existentes se utilizan durante un proyecto, la reducción del coste global puede ser importante. En efecto, los datos de la industria indican que el coste, el tiempo de adquisición y el número de defectos entregados al cliente se reducen.

<sup>7</sup> Otro hardware, *el entorno destino*, es la computadora sobre la que se va a ejecutar el software al entregarse al usuario final.

## 5.5 ESTIMACIÓN DEL PROYECTO DE SOFTWARE

Al principio, el coste del software constituía un pequeño porcentaje del coste total de los sistemas basados en computadora. Un error considerable en las estimaciones del coste del software tenía relativamente poco impacto. Hoy en día, el software es el elemento más caro de la mayoría de los sistemas informáticos. Para sistemas complejos, personalizados, un gran error en la estimación del coste puede ser lo que marque la diferencia entre beneficios y pérdidas. Sobre pasarse en el coste puede ser desastroso para el desarrollador.

La estimación del coste y del esfuerzo del software nunca será una ciencia exacta. Son demasiadas las variables —humanas, técnicas, de entorno, políticas— que pueden afectar al coste final del software y al esfuerzo aplicado para desarrollarlo. Sin embargo, la estimación del proyecto de software puede dejar de ser un oscuro arte para convertirse en una serie de pasos sistemáticos que proporcionen estimaciones con un grado de riesgo aceptable.



En una era de *outsourcing* y competencia creciente, la capacidad para estimar de un modo más exacto... se ha convertido en un factor crítico de supervivencia para muchos grupos TI.

Rob Thomssett

Para realizar estimaciones seguras de costes y esfuerzos tenemos varias opciones posibles:

1. Dejar la estimación para más adelante (obviamente, ¡podemos realizar una estimación al cien por cien fiable tras haber terminado el proyecto!).
2. Basar las estimaciones en proyectos similares ya terminados.
3. Utilizar «técnicas de descomposición» relativamente sencillas para generar las estimaciones de coste y de esfuerzo del proyecto.
4. Utilizar uno o más modelos empíricos para la estimación del coste y esfuerzo del software.

Desgraciadamente, la primera opción, aunque atractiva, no es práctica. Las estimaciones de costes han de ser proporcionadas *a priori*. Sin embargo, hay que reconocer que cuanto más tiempo esperemos, más cosas sabremos, y cuanto más sepamos, menor será la probabilidad de cometer serios errores en nuestras estimaciones.

La segunda opción puede funcionar razonablemente bien, si el proyecto actual es bastante similar a los esfuerzos pasados y si otras influencias del proyecto

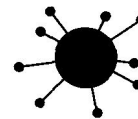
(por ejemplo: el cliente, las condiciones de gestión, el EIS [Entorno de Ingeniería del software], las fechas límites) son similares. Por desgracia, la experiencia anterior no ha sido siempre un buen indicador de futuros resultados.

Las opciones restantes son métodos viables para la estimación del proyecto de software. Desde un punto de vista ideal, se deben aplicar conjuntamente las técnicas indicadas; usando cada una de ellas como comprobación de las otras. Las *técnicas de descomposición* utilizan un enfoque de «divide y vencerás») para la estimación del proyecto de software. Mediante la descomposición del proyecto en sus funciones principales y en las tareas de ingeniería del software correspondientes, la estimación del coste y del esfuerzo puede realizarse de una forma escalonada idónea. Se pueden utilizar los *modelos empíricos de estimación* como complemento de las técnicas de descomposición, ofreciendo un enfoque de estimación potencialmente valioso por derecho propio. Cada modelo se basa en la experiencia (datos históricos) y toma como base:

$$d = f(v_i)$$

donde  $d$  es uno de los valores estimados (por ejemplo, esfuerzo, coste, duración del proyecto) y los  $v_i$ , son determinados parámetros independientes (por ejemplo, LDC o PF estimados).

Las *herramientas automáticas de estimación* implementan una o varias técnicas de descomposición o modelos empíricos. Cuando se combinan con una interfaz gráfica de usuario, las herramientas automáticas son una opción atractiva para la estimación. En sistemas de este tipo, se describen las características de la organización de desarrollo (por ejemplo, la experiencia, el entorno) y el software a desarrollar. De estos datos se obtienen las estimaciones de coste y de esfuerzo.



Herramientas CASE.

Cada una de las opciones viables para la estimación de costes del software, sólo será buena si los datos históricos que se utilizan como base de la estimación son buenos. Si no existen datos históricos, la estimación del coste descansará sobre una base muy inestable. En el Capítulo 4 examinamos las características de los datos de productividad del software y cómo pueden utilizarse como base histórica para la estimación.



## 5.6 TÉCNICAS DE DESCOMPOSICIÓN

La estimación de proyectos de software es una forma de resolución de problemas y, en la mayoría de los casos, el problema a resolver (es decir, desarrollar estimaciones de coste y de esfuerzo para un proyecto de software) es demasiado complejo para considerarlo como un todo. Por esta razón, descomponemos el problema, volviéndolo a definir como un conjunto de pequeños problemas (esperando que sean más manejables).

En el Capítulo 3, se estudió el enfoque de descomposición desde dos puntos de vista diferentes: descomposición del problema y descomposición del proceso. La estimación hace uso de una o ambas formas de particionamiento. Pero antes de hacer una estimación, el planificador del proyecto debe comprender el ámbito del software a construir y generar una estimación de su «tamaño».

### 5.6.1. Tamaño del software

La precisión de una estimación del proyecto de software se predice basándose en una serie de cosas: (1) el grado en el que el planificador ha estimado adecuadamente el tamaño del producto a construir; (2) la habilidad para traducir la estimación del tamaño en esfuerzo humano, tiempo y dinero (una función de la disponibilidad de métricas fiables de software de proyectos anteriores); (3) el grado en el que el plan del proyecto refleja las habilidades del equipo de software, y (4) la estabilidad de los requisitos del software y el entorno que soporta el esfuerzo de la ingeniería del software.

#### CUATRO CLAVE


El «tamaño» del software o construir puede estimarse utilizando una medida directa, LDC, o uno medido indirecto, PF.

En esta sección, consideramos el problema del *tamaño del software*. Puesto que una estimación del proyecto es tan buena como la estimación del tamaño del trabajo que va a llevarse a cabo, el tamaño representa el primer reto importante del planificador de proyectos. En el contexto de la planificación de proyectos, el tamaño se refiere a una producción cuantificable del proyecto de software. Si se toma un enfoque directo, el tamaño se puede medir en LDC. Si se selecciona un enfoque indirecto, el tamaño se representa como PF.

Putnam y Myers [PUT92] sugieren cuatro enfoques diferentes del problema del tamaño:

**Tamaño en «lógica difusa».** Este enfoque utiliza las técnicas aproximadas de razonamiento que son la piedra angular de la lógica difusa. Para aplicar este enfoque, el planificador debe identificar el tipo de aplicación, establecer su magnitud en una escala cuantitativa y refinar la magnitud dentro del rango original. Aunque se puede utilizar la experiencia personal, el planificador también debería tener acceso a una base de datos histórica de proyectos<sup>8</sup> para que las estimaciones se puedan comparar con la experiencia real.

**Tamaño en punto de función.** El planificador desarrolla estimaciones de características del dominio de información estudiadas en el Capítulo 4.

 ¿Cómo medimos el software que estamos planeando construir?

**Tamaño de componentes estándar.** El software se compone de un número de «componentes estándar» que son genéricos para un área en particular de la aplicación. Por ejemplo, los componentes estándar para un sistema de información son: subsistemas, módulos, pantallas, informes, programas interactivos, programas por lotes, archivos, LDC e instrucciones para objetos. El planificador de proyectos estima el número de incidencias de cada uno de los componentes estándar, y utiliza datos de proyectos históricos para determinar el tamaño de entrega por componente estándar. Para ilustrarlo, considere una aplicación de sistemas de información. El planificador estima que se generarán 18 informes. Los datos históricos indican que por informe se requieren 967 líneas de Cobol [PUT92]. Esto permite que el planificador estime que se requieren 17.000 LDC para el componente de informes. Para otros componentes estándar se hacen estimaciones y cálculos similares, y se producen resultados combinados de valores de tamaños (ajustados estadísticamente).

**Tamaño del cambio.** Este enfoque se utiliza cuando un proyecto comprende la utilización de software existente que se debe modificar de alguna manera como parte de un proyecto. El planificador estima el número y tipo (por ejemplo: reutilización, añadir código, cambiar código, suprimir código) de modificaciones que se deben llevar a cabo. Mediante una «proporción de esfuerzo» [PUT92] para cada tipo de cambio, se puede estimar el tamaño del cambio.


<sup>8</sup> Consulte la Sección 5.9 que trata brevemente las herramientas de estimación que utilizan una base de datos histórica y otras técnicas de tamaño estudiadas en esta sección.

Putnam y Myers sugieren que los resultados de cada uno de los métodos de tamaño señalados anteriormente se combinan estadísticamente para crear una *estimación de tres puntos o del valor esperado*. Esto se lleva a cabo desarrollando valores de tamaño optimistas (bajos), y más probables, y pesimistas (altos), y se combinan utilizando la Ecuación (5.1) que se describe en la sección siguiente.

### 5.6.2. Estimación basada en el problema

En el Capítulo 4, las líneas de código (LDC) y los puntos de función (PF) se describieron como medidas básicas a partir de las que se pueden calcular métricas de productividad. Los datos de LDC y PF se utilizan de dos formas durante la estimación del proyecto de software: (1) como una variable de estimación que se utiliza para «dimensionar» cada elemento del software, y (2) como métricas de línea base recopiladas de proyectos anteriores y utilizadas junto con variables de estimación para desarrollar proyecciones de coste y de esfuerzo.

Las estimaciones de LDC y PF son técnicas de estimación distintas. A pesar de que ambas tienen varias características en común. El planificador del proyecto comienza con un enfoque limitado para el ámbito del software y desde este estado intenta descomponer el software en funciones que se pueden estimar individualmente. Para cada función entonces se estiman las LDC y el PF (la variable de estimación). De forma alternativa, el planificador puede seleccionar otro componente para dimensionar clases u objetos, cambios o procesos de gestión en los que puede tener impacto.

 ¿Qué tienen en común la estimación orientada a PF y a LDC?

Las métricas de productividad de línea base (por ejemplo: LDC/pm o PF/pm<sup>9</sup>) se aplican entonces para la variable de estimación adecuada y se extrae el coste o el esfuerzo de la función. Las estimaciones de función se combinan para producir una estimación global del proyecto entero.



*Cuando se comparan métricas de productividad para proyectos, esté seguro de establecer una clasificación de los tipos de proyectos. Esto le permitirá calcular promedios para dominios específicos y la estimación será más exacta.*

Es importante señalar, sin embargo, que hay una sustancial diversidad en métricas de productividad, hacien-

do sospechar que se utilice únicamente una métrica de productividad de línea base. En general, el dominio del proyecto debería calcular las medias de LDC/pm o PF/pm. Es decir, los proyectos se deberían agrupar por tamaño de equipo, área de aplicación, complejidad y otros parámetros relevantes. Entonces se deberían calcular las medias del dominio local. Cuando se estima un proyecto nuevo, primero se debería asignar a un dominio, y a continuación utilizar la media del dominio adecuado para la productividad al generar la estimación. Las técnicas de estimación de LDC y PF difieren en el nivel de detalle que se requiere para la descomposición y el objetivo de la partición. Cuando se utiliza LDC como variable de estimación, la descomposición<sup>10</sup> es absolutamente esencial y con frecuencia se toman para considerables niveles de detalle. El enfoque de descomposición siguiente ha sido adaptado por Phillips [PHI98]<sup>11</sup>:

### CLAVE

Para estimaciones de LDC, la descomposición se centro en las funciones del software.

```

definir el ámbito del producto;
identificar funciones descomponiendo el ámbito;
hacer mientras haya funciones
    seleccionar una funciónj
    asignar todas las funciones a la lista de subfunciones;
    hacer mientras haya subfunciones
        seleccionar una subfunciónk
        si subfunciónk = subfunciónd descrita en una base
            de datos histórica
        entonces
            anotar datos históricos del coste,
            esfuerzo, tamaño, (LDC o PF) para
            la subfunciónd;
            ajustar datos históricos del coste,
            esfuerzo, tamaño basados en cual-
            quier diferencia;
            use datos del coste, esfuerzo, tama-
            ño ajustados para obtener una esti-
            mación parcial, Ep;
            estimación proyecto = suma de
            { Ep };
        sino
            si se puede estimar coste, esfuerzo,
            tamaño (LDC o PF) para subfunciónk
            entonces obtener estimación parcial,
            estimación proyecto = suma de
            { Ep };
            sino subdividir subfunciónk en sub-
            funciones más pequeñas;
            añadirlas a la lista de subfunciones;
        fin si
    fin si
fin hacer
fin hacer

```

<sup>9</sup> El acrónimo pm hace referencia a personas-mes

<sup>10</sup> En general se descomponen las funciones del problema. Sin embargo, se puede utilizar una lista de componentes estándar (Sección 5.6.1).

<sup>11</sup> El lenguaje informal de diseño del proceso señalado aquí se expone para ilustrar el enfoque general para el dimensionamiento. No tiene en cuenta ninguna eventualidad lógica.

El enfoque de descomposición visto anteriormente asume que todas las funciones pueden descomponerse en subfunciones que podrán asemejarse a entradas de una base de datos histórica. Si este no es el caso, deberá aplicarse otro enfoque de dimensionamiento. Cuanto más grande sea el grado de particionamiento, más probable será que pueda desarrollar estimaciones de LDC más exactas.

Para estimaciones de PF, la descomposición funciona de diferente manera. En lugar de centrarse en la función, se estiman cada una de las características del dominio de información —entradas, salidas, archivos de datos, peticiones, e interfaces externas— y los catorce valores de ajuste de la complejidad estudiados en el Capítulo 4. Las estimaciones resultantes se pueden utilizar para derivar un valor de PF que se pueda unir a datos pasados y utilizar para generar una estimación.

## CLAVE

Para estimaciones de PF, la descomposición se centra en las características del dominio de información.

Con independencia de la variable de estimación que se utilice, el planificador del proyecto comienza por estimar un rango de valores para cada función o valor del dominio de información. Mediante los datos históricos o (cuando todo lo demás falla) la intuición, el planificador estima un valor de tamaño optimista, más probable y pesimista para cada función, o cuenta para cada valor del dominio de información. Al especificar un rango de valores, se proporciona una indicación implícita del grado de incertidumbre.

## ¿Cómo calcular el valor esperado para el tamaño del software?

Entonces se calcula un valor de tres puntos o esperado. El *valor esperado* de la variable de estimación (tamaño), *VE*, se puede calcular como una media ponderada de las estimaciones optimistas ( $S_{opt}$ ), las más probables ( $S_m$ ), y las pesimistas ( $S_{pess}$ ). Por ejemplo,

$$VE = (S_{opt} + 4S_m + S_{pess}) / 6 \quad (5.1)$$

da crédito a la estimación «más probable» y sigue una distribución de probabilidad beta. Se asume que existe una probabilidad muy pequeña en donde el resultado del tamaño real quedará fuera de los valores pesimistas y optimistas.

Una vez que se ha determinado el valor esperado de la variable de estimación, se aplican datos históricos de productividad de LDC y PF. ¿Son correctas las estimaciones? La única respuesta razonable a esto es: «No podemos estar seguros». Cualquier técnica de estimación, no importa lo sofisticada que sea, se debe volver a comprobar con otro enfoque. Incluso entonces, va a prevalecer el sentido común y la experiencia.

## 5.6.3. Un ejemplo de estimación basada en LDC

Como ejemplo de técnicas de estimación de LDC y PF, tomemos en consideración un paquete de software a desarrollarse para una aplicación de diseño asistido por computadora (computer-aided design, CAD) de componentes mecánicos. Una revisión de la *especificación del sistema* indica que el software va a ejecutarse en una estación de trabajo de ingeniería y que debe interconectarse con varios periféricos de gráficos de computadora entre los que se incluyen un ratón, un digitalizador, una pantalla a color de alta resolución y una impresora láser.

Utilizando como guía una *especificación de sistema*, se puede desarrollar una descripción preliminar del ámbito del software:

El software de CAD aceptará datos geométricos de dos y tres dimensiones por parte del ingeniero. El ingeniero se interconectará y controlará el sistema de CAD por medio de una interfaz de usuario que va a exhibir las características de un buen diseño de interfaz hombre-máquina. Una base de datos de CAD contiene todos los datos geométricos y la información de soporte. Se desarrollarán módulos de análisis de diseño para producir la salida requerida que se va a visualizar en varios dispositivos gráficos. El software se diseñará para controlar e interconectar dispositivos periféricos entre los que se incluyen un ratón, un digitalizador, una impresora láser y un trazador gráfico (*plotter*).

La descripción anterior del ámbito es preliminar —no está limitada—. Para proporcionar detalles concretos y un límite cuantitativo se tendrían que ampliar todas las descripciones. Por ejemplo, antes de que la estimación pueda comenzar, el planificador debe determinar **qué** significa «características de un buen diseño de interfaz hombre-máquina» y cuál será el tamaño y la sofisticación de la «base de datos de CAD».

Para estos propósitos, se asume que ha habido más refinamiento y que se identifican las funciones de software siguientes:

- interfaz de usuario y facilidades de control (IUFC),
- análisis geométrico de dos dimensiones (AG2D),
- análisis geométrico de tres dimensiones (AG3D),
- gestión de base de datos (GBD),
- facilidades de presentación gráfica de computadora (FPGC),
- control de periféricos (CP),
- módulos de análisis del diseño (MAD).



Muchas aplicaciones modernas residen en una red o son parte de una arquitectura cliente/servidor. Por consiguiente, esté seguro que sus estimaciones contienen el esfuerzo requerido para el desarrollo del software de la «infraestructura».

Siguiendo la técnica de descomposición de LDC, se desarrolla la tabla de estimación mostrada en la Figura 5.3. Se desarrolla un rango de estimaciones de LDC para

cada función. Por ejemplo, el rango de estimaciones de LDC de la función del análisis geométrico de tres dimensiones es: optimista -4.600 LDC—; más probable -6.900—, y pesimista -8.600 LDC—. Aplicando la Ecuación (5.1), el valor esperado de la función del análisis geométrico es 6.800 LDC. Este número se introduce en la tabla. De forma similar se derivan otras estimaciones. Sumando en vertical la columna estimada de LDC, se establece una estimación de 33.200 líneas de código para el sistema de CAD.



*No sucumba o lo tentación de utilizar este resultado como estimación. Debería obtener otra estimación utilizando un método diferente.*

Una revisión de datos históricos indica que la productividad media de la organización para sistemas de este tipo es de 620 LDC/pm. Según una tarifa laboral de E8.000 por mes, el coste por línea de código es aproximadamente de £13,00. Según la estimación de LDC y los datos de productividad históricos, el coste total estimado del proyecto es de £431.000 y el esfuerzo estimado es de 54 personas-mes<sup>12</sup>.

Función	LDC estimada
Interface del usuario y facilidades de control (IUC)	2.300
Análisis geométrico de dos dimensiones (AG2D)	5.300
Análisis geométrico de tres dimensiones (AG3D)	6.800
Gestión de base de datos (GBD)	3.350
Facilidades de presentación gráfica de computadora (FPGC)	4.950
Control de periféricos (CP)	2.100
Módulos de análisis del diseño (MAD)	8.400
Líneas de códigos estimadas	33.200

FIGURA 5.3. Tabla de estimación para el método de LDC.

#### 5.6.4. Un ejemplo de estimación basada en PF

La descomposición de estimación basada en PF se centra en los valores de dominio de información, y no en las funciones del software. Teniendo en cuenta la tabla de cálculos de punto de función presentada en la Figura 5.4, el planificador del proyecto estima las entradas, salidas, peticiones, archivos e interfaces externas del software de CAD. Para los propósitos de esta estimación, el factor de ponderación de la complejidad se asume como la media. La Figura 5.4 presenta los resultados de esta estimación.



#### Referencia Web

Se puede obtener información sobre las herramientas de estimación del coste mediante PF en [www.spr.com](http://www.spr.com)

Valor de dominio de información	Opt.	Probable	Pes.	Cuenta est.	Peso	Cuenta PF
Número de entradas	20	24	30	24	4	97
Número de salidas	12	15	22	16	5	78
Número de peticiones	16	22	28	22	5	88
Número de archivos	4	4	5	4	10	42
Número de interfaces externas	2	2	3	2	7	15
<b>Cuenta total</b>						<b>320</b>

FIGURA 5.4. Estimación de los valores del dominio de información.

Como se describe en el Capítulo 4, se estima cada uno de los factores de ponderación de la complejidad, y se calcula el factor de ajuste de la complejidad:

Factor	Valor
Copia de seguridad y recuperación	4
Comunicaciones de datos	2
Proceso distribuido	0
Rendimiento crítico	4
Entorno operativo existente	3
Entrada de datos en línea (on-line)	4
Transacciones de entrada en múltiples pantallas	5
Archivos maestros actualizados en línea (on-line)	3
Complejidad de valores del dominio de información	5
Complejidad del procesamiento interno	5
Código diseñado para la reutilización	4
Conversión/instalación en diseño	3
Instalaciones múltiples	5
Aplicación diseñada para el cambio	5
Factor de ajuste de la complejidad	1,17

Finalmente, se obtiene el número estimado de PF:

$$PF_{\text{estimado}} = \text{cuenta total} \times [0,65 + 0,01 \times \sum(F_i)]$$

$$PF_{\text{estimado}} = 375$$

La productividad media organizacional para sistemas de este tipo es de 6,5 PF/pm. Según la tarifa laboral de E8.000 por mes, el coste por PF es aproximadamente de E1.230. Según la estimación de LDC y los datos de productividad históricos, el coste total estimado del proyecto es de £461.000, y el esfuerzo estimado es de 58 personas/mes.

<sup>12</sup> La estimación se redondea acercándose lo más posible a E1.000 y persona-mes.

### 5.6.5. Estimación basada en el proceso

La técnica más común para estimar un proyecto es basar la estimación en el proceso que se va a utilizar. Es decir, el proceso se descompone en un conjunto relativamente pequeño de actividades o tareas, y en el esfuerzo requerido para llevar a cabo la estimación de cada tarea.

Al igual que las técnicas basadas en problemas, la estimación basada en el proceso comienza con un esbozo de las funciones del software obtenidas a partir del ámbito del proyecto. Para cada función se debe llevar a cabo una serie de actividades del proceso de software. Las funciones y las actividades del proceso de software relacionadas se pueden representar como parte de una tabla similar a la presentada en la Figura 3.2.

#### Referencia cruzada

El marco de trabajo común del proceso (MCP) se estudia en el Capítulo 2.

Una vez que se mezclan las funciones del problema y las actividades del proceso, el planificador estima el esfuerzo (por ejemplo: personas-mes) que se requerirá para llevar a cabo cada una de las actividades del proceso de software en cada función. Estos datos constituyen la matriz central de la tabla de la Figura 3.2. Los índices de trabajo medios (es decir, esfuerzo coste/unidad) se aplican entonces al esfuerzo estimado a cada actividad del proceso. Es muy probable que en cada tarea varíe el índice de trabajo. El personal veterano se involucra de lleno con las primeras actividades y generalmente es mucho más caro que el personal junior, quienes están relacionados con las tareas de diseño posteriores, con la generación del código y con las pruebas iniciales.

Actividad →	CC	Planificación	Análisis de riesgo	Ingeniería		Construcción entrega		EC	Totales
Tarea →				Análisis	Diseño	Código	Prueba		
Función ↓									
IUFC				0.50	2.50	0.40	5.00	n/a	8.40
AG2D				0.75	4.00	0.60	2.00	n/a	7.35
AG3D				0.50	4.00	1.00	3.00	n/a	8.50
FPGC				0.50	3.00	1.00	1.50	n/a	6.00
GBD				0.50	3.00	0.75	1.50	n/a	5.75
CP				0.25	2.00	0.50	1.50	n/a	4.25
MAD				0.50	2.00	0.50	2.00	n/a	5.00
Totales	10.25	0.25	0.25	3.50	120.50	4.75	16.50		46.00
% Esfuerzo	1%	1%	1%	8%	45%	10%	36%		

CC = Comunicación cliente EC = Evaluación cliente

FIGURA 5.5. Tabla de estimación basada en el proceso.

Como último paso se calculan los costes y el esfuerzo de cada función, y la actividad del proceso de software. Si la estimación basada en el proceso se realiza independientemente de la estimación de LDC o PF, aho-

ra tendremos dos o tres estimaciones del coste y del esfuerzo que se pueden comparar y evaluar. Si ambos tipos de estimaciones muestran una concordancia razonable, hay una buena razón para creer que las estimaciones son fiables. Si, por otro lado, los resultados de estas técnicas de descomposición muestran poca concordancia, se debe realizar más investigación y análisis.

### 5.6.6. Un ejemplo de estimación basada en el proceso

Para ilustrar el uso de la estimación basada en el proceso, de nuevo consideremos el software de CAD presentado en la Sección 5.6.3. La configuración del sistema y todas las funciones del software permanecen iguales y están indicadas en el ámbito del proyecto.



Si el tiempo lo permite, realice una mayor descomposición al especificar las tareas; en la Figura 5.5, p. ej., se divide el análisis en sus tareas principales y se estima cada una por separado.

En la tabla de estimación de esfuerzos ya terminada, que se muestra en la Figura 5.5 aparecen las estimaciones de esfuerzo (en personas-mes) para cada actividad de ingeniería del software proporcionada para cada función del software de CAD (abreviada para mayor rapidez). Las actividades de *ingeniería* y de *construcción entrega* se subdividen en las tareas principales de ingeniería del software mostradas. Las estimaciones iniciales del esfuerzo se proporcionan para la *comunicación con el cliente*, *planificación* y *análisis de riesgos*. Estos se señalan en la fila **Total** en la parte inferior de la tabla. Los totales horizontales y verticales proporcionan una indicación del esfuerzo estimado que se requiere para el análisis, diseño, codificación y pruebas. Se debería señalar que el 53 por 100 de todo el esfuerzo se aplica en las tareas de ingeniería «frontales» (análisis y diseño de los requisitos) indicando la importancia relativa de este trabajo.

Basado en una tarifa laboral de £5.000 por mes, el coste total estimado del proyecto es de £230.000 y el esfuerzo estimado es de 46 personas/mes. A cada actividad de proceso del software o tareas de ingeniería del software se le podría asociar diferentes índices de trabajo y calcularlos por separado.

El esfuerzo total estimado para el software de CAD oscila de un índice bajo de 46 personas-mes (obtenido mediante un enfoque de estimación basado en el proceso) a un alto de 58 personas-mes (obtenido mediante un enfoque de estimación de PF). La estimación media (utilizando los tres enfoques) es de 53 personas-mes. La variación máxima de la estimación media es aproximadamente del 13 por ciento.

¿Qué ocurre cuando la concordancia entre las estimaciones es mala? Para responder a esta pregunta se ha



de reevaluar la información que se ha utilizado para hacer las estimaciones.

Muchas divergencias entre estimaciones se deben a una de dos causas:



*No espere que todas las estimaciones coincidan en uno o dos porcentajes. Si la estimación está dentro de una banda del 20 por 100, puede reconciliarse en un único valor.*

1. No se entiende adecuadamente el ámbito del proyecto o el planificador lo ha mal interpretado.
2. Los datos de productividad usados para técnicas de estimación basados en problemas no son apropiados para la aplicación, están obsoletos (ya no reflejan con precisión la organización de la ingeniería del software), o se han aplicado erróneamente.

El planificador debe determinar la causa de la divergencia y conciliar las estimaciones.

## 5.7 MODELOS EMPÍRICOS DE ESTIMACIÓN

Un *modelo de estimación* para el software de computadora utiliza fórmulas derivadas empíricamente para predecir el esfuerzo como una función de LDC o PF. Los valores para LDC o PF son estimados utilizando el enfoque descrito en las Secciones 5.6.2 y 5.6.3. Pero en lugar de utilizar las tablas descritas en esas secciones, los valores resultantes para LDC o PF se unen al modelo de estimación. Los datos empíricos que soportan la mayoría de los modelos de estimación se obtienen de una muestra limitada de proyectos. Por esta razón, el modelo de estimación no es adecuado para todas las clases de software y en todos los entornos de desarrollo. Por consiguiente, dos resultados obtenidos de dichos modelos se deben utilizar con prudencia<sup>13</sup>.

### 5.7.1. La estructura de los modelos de estimación



*Un modelo de estimación refleja la cantidad de proyectos a partir de donde se ha obtenido. Por consiguiente, el modelo es susceptible al dominio.*

Un modelo común de estimación se extrae utilizando el análisis de regresión sobre los datos recopilados de proyectos de software anteriores. La estructura global de tales modelos adquiere la forma de [MAT94]:

$$E = A + B \times (ev)^C \quad (5.2)$$

donde  $A$ ,  $B$  y  $C$  son constantes obtenidas empíricamente,  $E$  es el esfuerzo en personas-mes, y  $ev$  es la variable de estimación (de LDC o PF). Además de la relación

señalada en la Ecuación (5.2) la mayoría de los modelos de estimación tienen algún componente de ajuste del proyecto que permite ajustar  $E$  por otras características del proyecto (por ejemplo: complejidad del problema, experiencia del personal, entorno de desarrollo). Entre los muchos modelos de estimación orientados a LDC propuestos en la bibliografía se encuentran los siguientes:

$E = 5,2 \times (KLDC)^{0,91}$	Modelo de Walston-Felix
$E = 5,5 + 0,73 \times (KLDC)^{1,16}$	Modelo de Bailey-Basili
$E = 3,2 \times (KLDC)^{1,05}$	Modelo simple de Boehm
$E = 5,288 \times (KLDC)^{1,047}$	Modelo Doty para $KLDC > 9$

También se han propuesto los modelos orientados a PF. Entre estos modelos se incluyen:

$E = -13,39 + 0,0545 PF$	Modelo de Albretch y Gaffney
$E = 60,62 \times 7,728 \times 10^{-8} PF^3$	Modelo de Kemerer
$E = 585,7 + 15,12 PF$	Modelo de Matson, Bamett y Mellichamp

Un rápido examen de los modelos listados anteriormente indica que cada uno producirá un resultado diferente<sup>14</sup> para el mismo valor de LDC y PF. La implicación es clara. Los modelos de estimación se deben calibrar para necesidades locales.



*Ninguno de estos modelos debería ser aplicado inadecuadamente o su entorno.*

<sup>13</sup> En general se debería calibrar un modelo de estimación para las condiciones locales. El modelo se debería ejecutar utilizando los resultados de proyectos terminados. Los datos previstos por el modelo se deberían comparar con los resultados reales, y la eficacia del modelo (para condiciones locales) debería ser evaluada. Si la concordancia no es buena, los coeficientes del modelo se deben volver a calcular utilizando datos locales.

<sup>14</sup> Esta referencia se fundamenta en que estos modelos a menudo se derivan de un número relativamente pequeño de proyectos sólo en unos cuantos dominios de la aplicación.

### 5.7.2. El modelo COCOMO

Barry Boehm [BOE81], en su libro clásico sobre «economía de la ingeniería del software», introduce una jerarquía de modelos de estimación de software con el nombre de COCOMO, por *Constructive Cost Model* (Modelo Constructivo de Coste). El modelo COCOMO original se ha convertido en uno de los modelos de estimación de coste del software más utilizados y estudiados en la industria. El modelo original ha evolucionado a un modelo de estimación más completo llamado COCOMO II [BOE 96, BOE 00]. Al igual que su predecesor, COCOMO II es en realidad una jerarquía de modelos de estimación que tratan las áreas siguientes :

**Modelo de composición de aplicación.** Utilizado durante las primeras etapas de la ingeniería del software, donde el prototipado de las interfaces de usuario, la interacción del sistema y del software, la evaluación del rendimiento, y la evaluación de la madurez de la tecnología son de suma importancia.

**Modelo de fase de diseño previo.** Utilizado una vez que se han estabilizado los requisitos y que se ha establecido la arquitectura básica del software.

**Modelo de fase posterior a la arquitectura.** Utilizado durante la construcción del software.

Al igual que todos los modelos de estimación del software, el modelo COCOMO II descrito antes necesita información del tamaño. Dentro de la jerarquía del modelo hay tres opciones de tamaño distintas: puntos objeto, puntos de función, y líneas de código fuente.

El modelo de composición de aplicación COCOMO II utiliza los puntos objeto como se ilustra en los párrafos siguientes. Debería señalarse que también están disponibles otros modelos de estimación más sofisticados (utilizando PF y KLDC) que forman parte del COCOMO II.

#### Referencia Web

Se puede obtener información detallada sobre el COCOMO II, incluyendo software que puede descargar, en [sunset.usc.edu/COCOMOII/cocomo.html](http://sunset.usc.edu/COCOMOII/cocomo.html)

Del mismo modo que los puntos de función (Capítulo 4), el *punto objeto* es una medida indirecta de software que se calcula utilizando el total de (1) pantallas (de la interface de usuario), (2) informes, y (3) componentes que probablemente se necesiten para construir la aplicación. Cada instancia de objeto (por ejemplo,

una pantalla o informe) se clasifica en uno de los tres niveles de complejidad (esto es, básico, intermedio, o avanzado) utilizando los criterios sugeridos por Boehm [BOE96]. En esencia, la complejidad es una función del número y origen de las tablas de datos del cliente y servidor necesario para generar la pantalla o el informe y el número de vistas o secciones presentadas como parte de la pantalla o del informe.

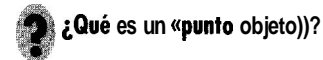
Tipo de objeto	Peso de la Complejidad		
	Básico	Intermedio	Avanzado
Pantalla	1	2	3
informe	2	5	a
Componente L3G			10

**TABLA 5.1.** Factores de peso de la complejidad para tipos de objeto [BOE96].

Una vez que se ha determinado la complejidad, se valora el número de pantallas, informes, y componentes de acuerdo con la Tabla 5.1. La cuenta de punto objeto se determina multiplicando el número original de instancias del objeto por el factor de peso de la tabla 5.1 y se suman para obtener la cuenta total de punto objeto. Cuando se va a aplicar el desarrollo basado en componentes o la reutilización de software en general, se estima el porcentaje de reutilización (%reutilización) y se ajusta la cuenta del punto objeto:

$$PON = (\text{puntos objeto}) \times [(100 - \%reutilización)/100]$$

donde PON significa «puntos objeto nuevos».



¿Qué es un «punto objeto»?

Para obtener una estimación del esfuerzo basada en el valor PON calculado, se debe calcular la «proporción de productividad». La Tabla 5.2 presenta la proporción de productividad para los diferentes niveles de experiencia del desarrollador y de madurez del entorno de desarrollo. Una vez determinada la proporción de productividad, se puede obtener una estimación del esfuerzo del proyecto:

$$\text{Esfuerzo estimado} = PON / PROD$$

En modelos COCOMO II más avanzados<sup>15</sup>, se requiere una variedad de factores de escala, conductores de coste y procedimientos de ajuste. Un estudio completo de estos temas están fuera del ámbito de este libro. El lector interesado debería consultar [BOE00] o visitar el sitio web COCOMO II.

<sup>15</sup> Como se señaló anteriormente, estos modelos hacen uso de las cuentas de PF y KLDC para la variable tamaño.

Experiencia/capacidad del desarrollador	Muy baja	Baja	Normal	Alta	Muy alta
Madurez/capacidad del entorno	Muy baja	Baja	Normal	Alta	Muy alta
PROD	4	7	13	25	50

**TABLA 5.2.** Proporciones de productividad para puntos objeto [BOE96].

### 5.7.3. La ecuación del software

La ecuación del software [PUT92] es un modelo multivariable dinámico que asume una distribución específica del esfuerzo a lo largo de la vida de un proyecto de desarrollo de software. El modelo se ha obtenido a partir de los datos de productividad para unos 4.000 proyectos actuales de software, un modelo de estimación tiene esta forma:

$$E = [LDC \times B^{0,333} / P]^3 \times (1/t^4) \quad (5.3)$$

donde

$E$  = esfuerzo en personas-mes o personas-año,

$t$  = duración del proyecto en meses o años,

$B$  = «factor especial de destrezas»<sup>16</sup>,

$P$  = «parámetro de productividad» que refleja:

- Madurez global del proceso y de las prácticas de gestión.
- La amplitud hasta donde se utilizan correctamente las normas de la ingeniería del software.
- El nivel de los lenguajes de programación utilizados. El estado del entorno del software.
- Las habilidades y la experiencia del equipo del software.
- La complejidad de la aplicación.

Los valores típicos para el desarrollo del software empotrado en tiempo real podrían ser  $P = 2.000$ ;  $P = 10.000$  para telecomunicaciones y software de sistemas; y  $P = 28.000$  para aplicaciones comerciales de

sistemas<sup>17</sup>. El parámetro de productividad se puede extraer para las condiciones locales mediante datos históricos recopilados de esfuerzos de desarrollo pasados.



### Referencia Web

Se puede obtener información de las herramientas de estimación del coste del software que se han desarrollado a partir de la ecuación del software en [www.qsm.com](http://www.qsm.com)

Es importante señalar que la ecuación del software tiene dos parámetros independientes: (1) una estimación del tamaño (en LDC) y (2) una indicación de la duración del proyecto en meses o años.

Para simplificar el proceso de estimación y utilizar una forma más común para su modelo de estimación, Putnam y Myers sugieren un conjunto de ecuaciones obtenidas de la ecuación del software. Un tiempo mínimo de desarrollo se define como:

$$t_{min} = 8,14 (LDC/P)^{0,43} \text{ en meses para } t_{min} > 6 \text{ meses} \quad (5.4a)$$

$$E = 180 B t^3 \text{ en personas-mes para } E \geq 20 \text{ personas-mes} \quad (5.4b)$$

Tenga en cuenta que  $t$  en la ecuación (5.4b) se representa en años.

Mediante las ecuaciones (5.4) con  $P = 12.000$  (valor recomendado para software científico) para el software de CAD estudiado anteriormente en este capítulo,

$$t_{min} = 8,14 (33.200 / 12.000)^{0,43}$$

$$t_{min} = 12,6 \text{ meses}$$

$$E = 180 \times 0,28 \times (1,05)^3$$

$$E = 58 \text{ personas-mes}$$

Los resultados de la ecuación del software se corresponde favorablemente con las estimaciones desarrolladas en la Sección 5.6. Del mismo modo que el modelo COCOMO expuesto en la sección anterior, la ecuación del software ha evolucionado durante la década pasada. Se puede encontrar un estudio de la versión extendida de este enfoque de estimación en [PUT97].

## 5.8 LA DECISIÓN DE DESARROLLAR-COMPRAR

En muchas áreas de aplicación del software, a menudo es más rentable adquirir el software de computadora que desarrollarlo. Los gestores de ingeniería del software se enfrentan con una decisión *desarrollar o comprar* que se puede complicar aún más con las opciones de adquisición: (1) el software se puede comprar (con licencia) ya desarrollado; (2) se pueden adquirir

componentes de software «totalmente experimentado» o «parcialmente experimentado» (Consulte la Sección 5.4.2), y entonces modificarse e integrarse para cumplir las necesidades específicas, o (3) el software puede ser construido de forma personalizada por una empresa externa para cumplir las especificaciones del comprador.

<sup>16</sup> B se incrementa lentamente a medida que crecen «la necesidad de integración, pruebas, garantía de calidad, documentación y habilidad de gestión» [PUT92]. Para programas pequeños (KLDC = 5 a 15),  $B = 0,16$ . Para programas mayores de 70 KLDC,  $B = 0,39$ .

<sup>17</sup> Es importante destacar que el parámetro de productividad puede obtenerse empíricamente de los datos locales de un proyecto.

Los pasos dados en la adquisición del software se definen según el sentido crítico del software que se va a comprar y el coste final. En algunos casos (por ejemplo: software para PC de bajo coste) es menos caro comprar y experimentar que llevar a cabo una larga evaluación de potenciales paquetes de software. Para productos de software más caros, se pueden aplicar las directrices siguientes:

1. Desarrollo de una especificación para la función y rendimiento del software deseado. Definición de las características medibles, siempre que sea posible.



*Hoy veces en las que el software proporciona una solución «perfecta» excepto por algunos aspectos especiales de los que puede prescindir. En muchos casos, ¡merece la pena vivir sin éstos!*

2. Estimación del coste interno de desarrollo y la fecha de entrega.
- 3a. Selección de tres o cuatro aplicaciones candidatos que cumplan mejor las especificaciones.
- 3b. Selección de componentes de software reutilizables que ayudarán en la construcción de la aplicación requerida.
4. Desarrollo de una matriz de comparación que presente la comparación una a una de las funciones clave. Alternativamente, realizar el seguimiento de las pruebas de evaluación para comparar el software candidato.
5. Evaluación de cada paquete de software o componente según la calidad de productos anteriores, soporte del vendedor, dirección del producto, reputación, etc.
6. Contacto con otros usuarios de dicho software y petición de opiniones.

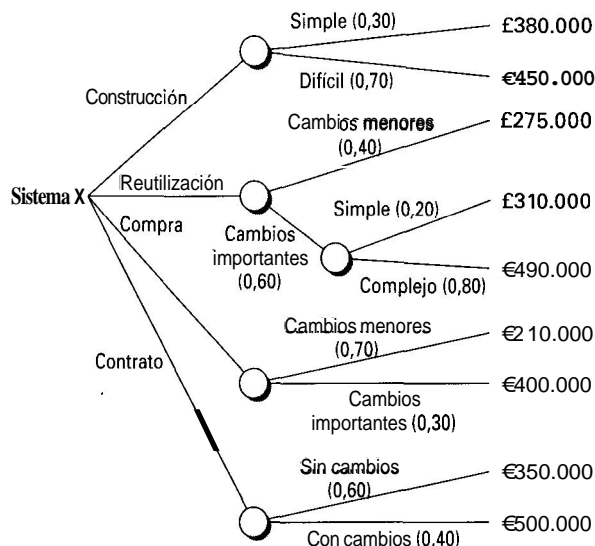


FIGURA 5.6. Árbol de decisión para apoyar la decisión desarrollar-comprar.

En el análisis final, la decisión de desarrollar—comprar se basa en las condiciones siguientes: (1) ¿La fecha de entrega del producto de software será anterior que la del software desarrollado internamente? (2) ¿Será el coste de adquisición junto con el de personalización menor que el coste de desarrollo interno del software? (3) ¿Será el coste del soporte externo (por ejemplo: un contrato de mantenimiento) menor que el coste del soporte interno? Estas condiciones se aplican a cada una de las opciones señaladas anteriormente.

### 5.8.1. Creación de un árbol de decisiones

Los pasos descritos anteriormente se pueden especificar mediante técnicas estadísticas tales como el *análisis del árbol de decisión* [BOE89]. Por ejemplo, la Figura 5.6 representa un árbol de decisión para un sistema X basado en software. En este caso, la organización de la ingeniería del software puede (1) construir el sistema X desde el principio; (2) reutilizar los componentes existentes de «experiencia parcial» para construir el sistema; (3) comprar un producto de software disponible y modificarlo para cumplir las necesidades locales; o (4) contratar el desarrollo del software a un vendedor externo.



**¿Existe un método sistemático para ordenar las opciones relacionadas con la decisión desarrollar-comprar?**

Si se va a construir un sistema desde el principio, existe una probabilidad del 70 por 100 de que el trabajo sea difícil. Mediante las técnicas de estimación estudiadas en este capítulo, el planificador del proyecto estima que un esfuerzo de desarrollo difícil costará £450.000. Un esfuerzo de desarrollo «simple» se estima que cueste £380.000. El valor esperado del coste, calculado a lo largo de la rama del árbol de decisión es:

$$\text{coste esperado} = \sum (\text{probabilidad del camino})_i \times (\text{coste estimado del camino})_i$$

donde  $i$  es el camino del árbol de decisión. Para el camino construir,

$$\text{coste esperado}_{\text{construcción}} = 0,30 (£380K) + 0,70 (£450K) = £429K$$

También se muestran los otros caminos del árbol de decisión, los costes proyectados para la reutilización, compra y contrato, bajo diferentes circunstancias. Los costes esperados de estas rutas son:

$$\text{coste esperado}_{\text{reutilización}} = 0,40 (£275K) + 0,60 [0,20 (£310K) + 0,80 (£490K)] = £382K$$

$$\text{coste esperado}_{\text{compra}} = 0,70 (£210K) + 0,30 (£400K) = £267K$$

$$\text{coste esperado}_{\text{contrato}} = 0,60 (£350K) + 0,40 (£500K) = £410K$$

Según la probabilidad y los costes proyectados que se han señalado en la Figura 5.6, el coste más bajo esperado es la opción «compra».



Se puede encontrar un tutorial excelente sobre el análisis del árbol de decisión en [www.demon.co.uk/mindtool/dectree.html](http://www.demon.co.uk/mindtool/dectree.html)

Sin embargo, es importante señalar que se deben considerar muchos criterios —no sólo el coste— durante el proceso de toma de decisiones. La disponibilidad, la experiencia del desarrollador/vendedor/contratante, la conformidad con los requisitos, la política «local», y la probabilidad de cambios son sólo uno de los pocos criterios que pueden afectar la última decisión para construir, volver a utilizar o contratar.

### 5.8.2. Subcontratación (outsourcing)

Más pronto o más tarde toda compañía que desarrolla software de computadora hace una pregunta fundamental: «¿Existe alguna forma por la que podamos conseguir a bajo precio el software y los sistemas que necesitamos?» La respuesta a esta pregunta no es simple, y las discusiones sobre esta cuestión dan como respuesta una simple palabra: subcontratación (outsourcing).



Se puede encontrar información útil (documentos, enlaces) acerca del outsourcing en [www.outsourcing.com](http://www.outsourcing.com)

El concepto outsourcing es extremadamente simple. Las actividades de ingeniería del software se contratan

con un tercero, quien hace el trabajo a bajo coste asegurando una alta calidad. El trabajo de software llevado dentro de la compañía se reduce a una actividad de gestión de contratos.



A modo de regla, el outsourcing requiere incluso más gestión experta que el desarrollo interno.

Steve McConnell

La decisión de contratar fuentes externas puede ser estratégica o táctica. En el nivel estratégico, los gestores tienen en consideración si una parte importante de todo el trabajo del software puede ser contratado a otros. En el nivel táctico, un jefe de proyecto determina si algunas partes o todo el proyecto es aconsejable realizarlo mediante subcontratación.

Con independencia de la amplitud del enfoque, la decisión de elegir outsourcing es a menudo financiera. Un estudio detallado del análisis financiero de la decisión del outsourcing está fuera del ámbito de este libro y se reserva mejor para otros (por ejemplo: [MIN95]). Sin embargo, merece la pena realizar un repaso de los pros y los contras de la decisión.

En el lado positivo, los ahorros de coste se pueden lograr reduciendo el número de personas y las instalaciones (por ejemplo: computadoras, infraestructura) que los apoyan. En el lado negativo, una compañía pierde control sobre el software que necesita. Como el software es una tecnología que diferencia sus sistemas, servicios, y productos, una compañía corre el riesgo de poner su destino en manos de un tercero.

## 5.9 HERRAMIENTAS AUTOMÁTICAS DE ESTIMACIÓN

Las técnicas de descomposición y los modelos empíricos de estimación descritos en las secciones anteriores se pueden implementar con software. Las herramientas automáticas de estimación permiten al planificador estimar costes y esfuerzos, así como llevar a cabo análisis del tipo «qué pasa si» con importantes variables del proyecto, tales como la fecha de entrega o la selección de personal. Aunque existen muchas herramientas automáticas de estimación, todas exhiben las mismas características generales y todas realizan las seis funciones genéricas mostradas a continuación [JON96]:

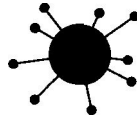
- 1. Dimensionamiento de las entregas del proyecto.** Se estima el «tamaño» de uno o más productos de software. Los productos incluyen la representación externa del software (por ejemplo, pantallas, informes), el software en sí (por ejemplo, KLDC), su funcionalidad (por ejemplo, puntos de función), y la información descriptiva (por ejemplo, documentos).
- 2. Selección de las actividades del proyecto.** Se selecciona el marco de trabajo del proceso adecuado

(Capítulo 2) y se especifica el conjunto de tareas de ingeniería del software.

- 3. Predicción de los niveles de la plantilla.** Se especifica el número de personas disponibles para realizar el trabajo. Esto es muy importante, puesto que la relación entre las personas disponibles y el trabajo (esfuerzo previsto) no es muy lineal.
- 4. Predicción del esfuerzo del software.** Las herramientas de estimación utilizan uno o más modelos (por ejemplo, Sección 5.7) que relacionan el tamaño de las entregas del proyecto con el esfuerzo necesario para producirlas.
- 5. Predicción del coste del software.** Dado los resultados del paso 4, los costes pueden calcularse asignando proporciones del trabajo a las actividades del proyecto señaladas en el paso 2.
- 6. Predicción de la planificación del software.** Cuando se conoce el esfuerzo, los niveles de la plantilla y las actividades del proyecto, se puede realizar un borrador de la planificación asignando el trabajo a través



de actividades de ingeniería del software basadas en modelos recomendados para la distribución del esfuerzo (Capítulo 7).



Herramientas Case.

Cuando se aplican distintas herramientas de estimación a los mismos datos del proyecto, se observa una variación relativamente grande entre los resultados estimados. Pero lo que es más importante, a veces los valores son bastante diferentes de los valores reales. Esto refuerza la idea de que los resultados obtenidos por las herramientas de estimación se deben usar sólo como «punto de partida» para la obtención de estimaciones —no como única fuente para la estimación—.

## RESUMEN

El planificador del proyecto de software tiene que estimar tres cosas antes de que comience el proyecto: cuánto durará, cuánto esfuerzo requerirá y cuánta gente estará implicada. Además, el planificador debe predecir los recursos (de hardware y de software) que va a requerir, y el riesgo implicado.

El enunciado del ámbito ayuda a desarrollar estimaciones mediante una o varias de las técnicas siguientes: descomposición, modelos empíricos y herramientas automáticas. Las técnicas de descomposición requieren un esbozo de las principales funciones del software, seguido de estimaciones (1) del número de LDC's, (2) de los valores seleccionados dentro del dominio de la información, (3) del número de personas-mes requeridas para implementar cada función, o (4) del número

de personas-mes requeridas para cada actividad de ingeniería del software. Las técnicas empíricas usan expresiones empíricamente obtenidas para el esfuerzo y para el tiempo, con las que se predicen esas magnitudes del proyecto. Las herramientas automáticas implementan un determinado modelo empírico.

Para obtener estimaciones exactas para un proyecto, generalmente se utilizan al menos dos de las tres técnicas referidas anteriormente. Mediante la comparación y la conciliación de las estimaciones obtenidas con las diferentes técnicas, el planificador puede obtener una estimación más exacta. La estimación del proyecto de software nunca será una ciencia exacta, pero la combinación de buenos datos históricos y de técnicas sistemáticas pueden mejorar la precisión de la estimación.

## REFERENCIAS

- [BEN92] Bennatan, E. M., *Software Project Management: A Practitioner's Approach*, McGraw-Hill, 1992.
- [BOE81] Boehm, B., *Software Engineering Economics*, Prentice-Hall, 1981.
- [BOE89] Boehm, B., *Risk Management*, IEEE Computer Society Press, 1989.
- [BOE96] Boehm, B., «Anchoring the Software Process», *IEEE Software*, vol. 13, n.º 4, Julio 1996, pp. 73-82.
- [BOE00] Boehm, B., et al., *Software Cost Estimation in COCOMO II*, Prentice Hall, 2000.
- [BRO75] Brooks, F., *The Mythical Man-Month*, Addison-Wesley, 1975.
- [GAU89] Gause, D. C., y G. M. Weinberg, *Exploring Requirements: Quality Before Design*, Dorset House, 1989.
- [HOO91] Hooper, J. W. E., y R. O. Chester, *Software Reuse: Guidelines and Methods*, Plenum Press, 1991.
- [JON96] Jones, C., «How Software Estimation Tools Work», *American Programmer*, vol. 9, n.º 7, Julio 1996, pp. 19-27.
- [MAH96] Mah, M., *Quantitative Software Management*, Inc., private communication.
- [MAT94] Matson, J., B. Barrett y J. Mellichamp, «Software Development Cost Estimation Using Function Points», *IEEE Trans. Software Engineering*, vol. 20, n.º 4, Abril 1994, pp. 275-287.
- [MIN95] Minoli, D., *Analyzing Outsourcing*, McGraw-Hill, 1995.
- [PHI98] Phillips, D., *The Software Project Manager's Handbook*, IEEE Computer Society Press, 1998.
- [PUT92] Putnam, L., y W. Myers, *Measures for Excellence*, Yourdon Press, 1992.
- [PUT97a] Putnam, L., y W. Myers, «How Solved is the Cost Estimation Problem», *IEEE Software*, Noviembre 1997, pp. 105-107.
- [PUT97b] Putnam, L., y W. Myers, *Industrial Strength Software: Effective Management Using Measurement*, IEEE Computer Society Press, 1997.

## PROBLEMAS Y PUNTOS A CONSIDERAR

**5.1.** Suponga que es el gestor de proyectos de una compañía que construye software para productos de consumo. Ha contratado una construcción de software para un sistema de seguridad del hogar. Escriba una especificación del ámbito que describa el software. Asegúrese de que su enunciado del ámbito sea limitado. Si no está familiarizado con sistemas de seguridad del hogar, investigue un poco antes de comenzar a escribir. *Alternativa:* sustituya el sistema de seguridad del hogar por otro problema que le sea de interés.

**5.2.** La complejidad del proyecto de software se trata brevemente en la Sección 5.1. Desarrolle una lista de las características de software (por ejemplo: operación concurrente, salida gráfica, etc.), que afecte a la complejidad de un proyecto. Dé prioridad a la lista.

**5.3.** El rendimiento es una consideración importante durante la planificación. Discuta si puede interpretar el rendimiento de otra manera, dependiendo del área de aplicación del software.

**5.4.** Haga una descomposición de las funciones software para el sistema de seguridad del hogar descrita en el Problema 5.1. Estime el tamaño de cada función en LDC. **Asumiendo** que su organización produce 450LDC/pm con una tarifa laboral de \$7.000 por persona-mes, estime el esfuerzo y el coste requerido para construir el software utilizando la técnica de estimación basada en LDC que se describe en la Sección 5.6.3.

**5.5.** Utilizando las medidas de punto de función de tres dimensiones que se describe en el Capítulo 4, calcule el número de PF para el software del sistema de seguridad del hogar, y extraiga las estimaciones del esfuerzo y del coste mediante la técnica de estimación basada en PF que se describe en la Sección 5.6.4.

**5.6.** Utilice el modelo COCOMO II para estimar el esfuerzo necesario para construir software para un simple ATM que produzca 12 pantallas; 10 informes, y que necesite aproxima-

damente 80 componentes de software. Asuma complejidad «media» y maduración desarrollador/entorno media. Utilice el modelo de composición de aplicación con puntos objeto.

**5.7.** Utilice la «ecuación del software» para estimar el software del sistema de seguridad del hogar. Suponga que las Ecuaciones (5.4) son aplicables y que  $P = 8.000$ .

**5.8.** Compare las estimaciones de esfuerzo obtenidas de los Problemas 5.5 y 5.7. Desarrolle una estimación simple para el proyecto mediante una estimación de tres puntos. ¿Cuál es la desviación estándar?, y ¿cómo afecta a su grado de certeza sobre la estimación?

**5.9.** Mediante los resultados obtenidos del Problema 5.8, determine si es razonable esperar que el resultado se pueda construir dentro de los seis meses siguientes y cuántas personas se tendrían que utilizar para terminar el trabajo.

**5.10.** Desarrolle un modelo de hoja de cálculo que implemente una técnica de estimación o más, descritas en el capítulo. Alternativamente, obtenga uno o más modelos directos para la estimación desde la web.

**5.11.** *Para un equipo de proyecto:* Desarrolle una herramienta de software que implemente cada una de las técnicas de estimación desarrolladas en este capítulo.

**5.12.** Parece extraño que las estimaciones de costes y planificaciones temporales se desarrollen durante la planificación del proyecto de software —antes de haber realizado un análisis de requisitos o un diseño detallado—. ¿Por qué cree que se hace así? ¿Existen circunstancias en las que no deba procederse de esta forma?

**5.13.** Vuelva a calcular los valores esperados señalados en el árbol de decisión de la Figura 5.6 suponiendo que todas las ramas tienen una probabilidad de 50-50. ¿Por qué cambia esto su decisión final?

## OTRAS LECTURAS Y FUENTES DE INFORMACIÓN

La mayoría de los libros de gestión de proyectos de software contienen estudios sobre la estimación de proyectos. Jones (*Estimating Software Costs*, McGraw Hill, 1998) ha escrito el tratado más extenso sobre la estimación de proyectos publicado hasta la fecha. Su libro contiene modelos y datos aplicables a la estimación del software en todo el dominio de la aplicación. Roetzheim y Beasley (*Software Project Cost and Schedule Estimating: Best Practices*, Prentice-Hall, 1997) presentan varios modelos útiles y sugieren las directrices paso a paso para generar las estimaciones mejores posibles.

Los libros de Phillips [PHI98], Bennatan (*On Time, Within Budget: Software Project Management Practices and Techniques*, Wiley, 1995), Whitten (*Managing Software Development Projects: Formula for Success*, Wiley, 1995), Wellman (*Software Costing*, Prentice-Hall, 1992), Londeix (*Cost Estimation for Software Development*, Addison-Wesley, 1987) contienen información útil sobre la planificación y la estimación de proyectos de software.

El tratamiento detallado de la estimación del coste de software de Putnam y Myers ([PUT92] y [PUT97b]), y el libro de Boehm sobre la economía de la ingeniería del software ([BOE81] y COCOMO II [BOE00]) describen los modelos de estimación

empíricos para la estimación del software. Estos libros proporcionan un análisis detallado de datos que provienen de cientos de proyectos de software. Un libro excelente de DeMarco (*Controlling Software Projects*, Yourdon Press, 1982) proporciona una profunda y valiosa visión de la gestión, medición y estimación de proyectos de software. Sneed (*Software Engineering Management*, Wiley, 1989) y Macro (*Software Engineering: Concepts and Management*, Prentice—Hall, 1990) estudian la estimación del proyecto de software con gran detalle.

La estimación del coste de líneas de código es el enfoque más comúnmente utilizado. Sin embargo, el impacto del paradigma orientado a objetos (consulte la Parte Cuarta) puede invalidar algunos modelos de estimación. Lorenz y Kidd (*Object — Oriented Software Metrics*, Prentice—Hall, 1994) y Cockburn (*Surviving Object — Oriented Projects*, Addison—Wesley, 1998) estudian la estimación para sistemas orientados a objetos.

En Internet están disponibles una gran variedad de fuentes de información sobre la planificación y estimación del software. Se puede encontrar una lista actualizada con referencias a sitios (páginas) web que son relevantes para la estimación del software en <http://www.pressman5.com>.